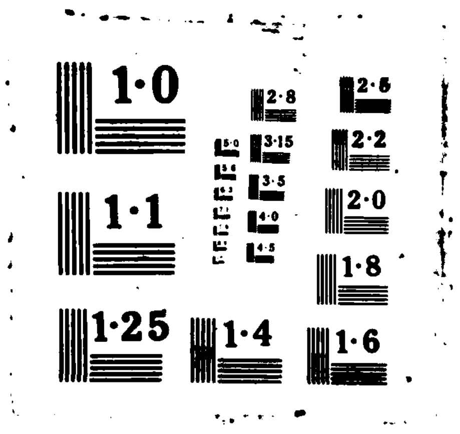
AD-A177 792 PARALLEL IMPLEMENTATIONS OF GRADIENT BASED ITERATIVE ALGORITHMS FOR A CLA.. (U) JOHNS HOPKINS UNIV BALTIMORE MD DEPT OF ELECTRICAL ENGINEERIN.. G G MEYER ET AL. 1/1

UNCLASSIFIED 28 FEB 87 JHU/ECE-87/03 AFOSR-85-0097 F/G 9/2 NL

END
4 87
DTIC


1·0
2·8
2·5
3·2
3·15
2·2
3·6
3·5
2·0
4·0
4·5
1·1
1·8
1·25
1·4
1·6

(12)

PARALLEL IMPLEMENTATIONS OF GRADIENT BASED ITERATIVE ALGORITHMS FOR A CLASS OF DISCRETE OPTIMAL CONTROL PROBLEMS

Gerard G. L. Meyer and Louis J. Podrazik

REPORT JHU/ECE-87/03

Electrical and Computer Engineering Department
The Johns Hopkins University
Baltimore, Maryland 21218

This work was supported by the Air Force Office of Scientific Research under Contract AFOSR-85-0097.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A177792

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Unrestricted
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) JHU/ECE-87/03	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The Johns Hopkins University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research /NM
6c. ADDRESS (City, State and ZIP Code) Charles and 34th Streets Baltimore, Maryland 21218	7b. ADDRESS (City, State and ZIP Code) Bolling AFB, Washington DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR/PKZ 8b. OFFICE SYMBOL (If applicable) N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-85-0097
8c. ADDRESS (City, State and ZIP Code) Building 410 Bolling AFB - DC 20332-6448	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT NO.

11. TITLE (Include Security Classification) Parallel Implementations of Gradient Based Iterative Algorithms For a Class of Discrete Optimal Control Problems

12. PERSONAL AUTHOR(S)
Gerard G. L. Meyer and Louis J. Podrazik

13a. TYPE OF REPORT Interim	13b. TIME COVERED FROM 1/1/87 TO 2/28/87	14. DATE OF REPORT (Yr., Mo., Day) February 28, 1987	15. PAGE COUNT 29

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Algorithm, parallelism, optimal control, gradient procedure

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper we present the parallel implementations of two iterative gradient based algorithms to solve the unconstrained linear quadratic regulator optimal control problem. We show that parallel evaluation of the step length and gradient of the quadratic cost function can be efficiently performed as a function of the number of processors. We then embed our parallel step length and gradient procedures to produce parallel implementations of the gradient and conjugate gradient methods that may be executed on an SIMD machine.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED [✓] SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Major Brian Woodruff	22b. TELEPHONE NUMBER (Include Area Code) (202)-767-5027 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE. Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

ABSTRACT

~~In~~ this paper ~~we~~ present the parallel implementations of two iterative gradient based algorithms to solve the unconstrained linear quadratic regulator optimal control problem. ~~We~~ show that parallel evaluation of the step length and gradient of the quadratic cost function can be efficiently performed as a function of the number of processors. We then embed our parallel step length and gradient procedures to produce parallel implementations of the gradient and conjugate gradient methods that may be executed on an SIMD machine.

I. INTRODUCTION

Previous parallel approaches to the solution of optimal control problems [LAR73], [MEY73], [SCH81], [TRA80] have been devised without explicitly taking into consideration the computational environment. In particular, when the number of available processors is small in relation to the problem size, the above techniques simply fold the computations to fit the number of processors. More efficient parallel algorithms may be devised by considering the computational environment throughout the algorithm synthesis. Toward that end, we present in this paper a parallel procedure for gradient evaluation which is formulated as a function of the number of available processors. Although presented in the context of unconstrained optimal control, our results for gradient computation are also applicable to constrained problems. Furthermore, we show that the step size obtained as a result of the line search performed at each iteration may also be efficiently computed in parallel. We then combine the techniques for parallel gradient evaluation and step size determination to produce parallel implementations of the best-step steepest descent method and the Fletcher-Reeves conjugate gradient method to solve the linear quadratic regulator (LQR) control problem [LEW86].

It is well known that a closed loop feedback form solution exists for the LQR problem. Our motivation for solving that problem using iterative gradient based techniques is that our basic parallel approach can be applied to more complex control problems in which the system dynamics can be linearized and the cost approximated quadratically. Furthermore, efficient parallel implementations of gradient methods such as best-step steepest descent and conjugate gradient suggests that similar parallel implementations of penalty function and gradient projection methods may be used to solve constrained control problems.

Our approach to the parallel evaluation of the step and gradient reduces the total number of operations required by sharing common terms when possible and then introduces parallelism. The degree of parallelism exhibited by the step and gradient computation techniques presented in this paper varies as a function of the number of processors to be used. We constrain the number of available processors, p, to lie in the range $1 \leq p \leq nN^{½}$, where n is the size of the system state vector, N is the number of stages in the control process and we assume $n \geq m$, where m is the size of the control. One of the features of the proposed parallel iterative algorithms is that their structure is completely specified by the number of processors whenever the number of stages $N \geq \left(\frac{p}{n}\right)^2$.

An efficient technique for gradient evaluation using a single processor has been discussed by Polak [POL71, pp.66-69]. A direct implementation of this technique on p processors achieves linear speedup for p up to n; however, for $p > n$, the speedup is significantly reduced. In this paper, we present an approach to gradient computation that may be efficiently implemented on up to $nN^{½}$ processors and reduces to a direct parallel implementation of the gradient evaluation given in [POL71] when $1 \leq p \leq n$. However, for $n < p \leq nN^{½}$, we show that our approach achieves speedup greater than $\frac{1}{2}(p + n)$.

A critical step in our approach involves the parallel computation of the state and costate vectors. When $n = 1$, the computations reduce to solving a forward linear recurrence system followed by a reverse linear recurrence system, both of size N. The parallel evaluation of m-th order linear recurrence systems has been extensively studied [KOG73], [CHE75], [KUC76], [SAM75], [SAM77], [CHE78], [GAJ81], [CAR84], [MEY85] and [MEY86]. To solve first-order linear block recurrence systems in parallel, we use a blocked formulation of the approach presented in [MEY86], [MEY86]. In addition to requiring less steps to solve a first-order system than any of the above when $1 \leq p \leq nN^{1/2}$, the approach in [MEY86] can be implemented on a simple ring network with broadcasting capability.

The organization of this paper is as follows : in Section II, we state the unconstrained discrete linear quadratic optimal control problem, examine the gradient of the cost function and give the steepest descent algorithm we shall consider. Section III presents the step length and gradient computations required at each iteration. In Section IV we give parallel procedures to solve the linear recurrence systems required by Section III. Based upon the results of Sections III and IV, Section V presents parallel implementations of the best-step steepest descent method to solve the LQR problem and the corresponding performance analysis. Finally, in Section VI conclusions are presented.

II. PRELIMINARIES

We consider the LQR discrete optimal control problem:

Problem 1: Given an m-input, discrete, time-varying linear system in which we are given the initial state, z_0, and

$$z_i = A_i z_{i-1} + B_i u_i , \quad i = 1,2,...N, \tag{1}$$

where for $i = 0,1,...,N$, z_i in E^n is the state of the system at time i and for $i = 1,2,...,N$, u_i in E^m is the control at time i, find the mN control vector $u = (u_1^t, u_2^t, \ldots, u_N^t)^t$ that minimizes the performance index

$$J(u) = \frac{1}{2} \left(z^t Qz + u^t Ru \right),$$

where z is the nN vector $z = (z_1^t, z_2^t, \ldots, z_N^t)^t$, $Q = Diag(Q_1, Q_2,...,$

Q_N) is the $nN \times nN$ block matrix that consists of N $n \times n$ symmetric positive semi-definite blocks Q_i and $R = Diag(R_1, R_2, \ldots, R_N)$ is the $mN \times mN$ block matrix that consists of N $m \times m$ symmetric positive definite blocks R_i.

The hypotheses on the matrices Q and R insure that Problem 1 possesses a unique solution u^* [LUE84] and that $\left(\frac{dJ}{du}\right)_{u=u^*} = 0$

We now introduce a formulation for $\frac{dJ}{du}$ that is used in our parallel implementation of gradient algorithms.

A direct application of the chain rule for differentiation [GRA81] yields

$$\frac{dJ}{du} = \frac{\partial J}{\partial u} + \frac{\partial J}{\partial z}\frac{dz}{du}, \tag{2}$$

where the z_i's are determined in accordance with Eq. (1), $\frac{\partial J}{\partial u}$ and $\frac{\partial J}{\partial z}$ are the $1 \times mN$ and $1 \times nN$ Jacobian matrices $u^t R$ and $z^t Q$ respectively, and $\frac{dz}{du}$ is the $nN \times mN$ block lower triangular Jacobian matrix that consists of N^2 $n \times m$ blocks $\left(\frac{dz}{du}\right)_{ij} = \frac{dz_i}{du_j}$ obtained by the chain rule for all i and j in $[1,2,\ldots,N]$ by

$$\frac{dz_i}{du_j} = \begin{cases} 0 & \text{if } i < j \\ B_j & \text{if } i = j \\ A_i A_{i-1} \cdots A_{j+1} B_j & \text{if } i > j. \end{cases} \tag{3}$$

Eq. (3) shows that the influence matrix $\frac{dz}{du}$ satisfies $F_z \frac{dz}{du} = F_u$, where F_z is the $nN \times nN$ block lower bidiagonal matrix that consists of N^2 $n \times n$ blocks $\left(F_z\right)_{ij}$ defined for all i and j in $[1,2,\ldots,N]$ by

$$\left(F_z\right)_{ij} = \begin{cases} I & \text{if } i = j \\ -A_i & \text{if } i = j+1 \\ 0 & \textit{otherwise} \end{cases}$$

and F_u is the $nN \times mN$ block diagonal matrix that consists of N^2 $n \times m$ blocks $\left(F_u\right)_{ij}$ defined for all i and j in $[1,2,\ldots,N]$ by

$$\left(F_u\right)_{ij} = \begin{cases} B_i & \text{if } i = j \\ 0 & \textit{otherwise}. \end{cases}$$

Let $g = \frac{dJ^t}{du}$ be the the $mN \times 1$ gradient of $J(u)$ with respect to u. The matrix F_z is non-singular and thus we may rewrite Eq. (2) as

$$g = Ru + F_u^t F_z^{-t} Qz ,$$

where z again satisfies Eq. (1).

Let F_0 be the $nN \times n$ block matrix that consists of N $n \times n$ blocks $\left(F_0\right)_i$ defined for all i in $[1,2,\ldots,N]$ by

$$\left(F_0\right)_i = \begin{cases} A_1 & \text{if } i = 1 \\ 0 & \textit{otherwise} , \end{cases}$$

and let λ be the nN costate vector $\lambda = (\lambda_1^t, \lambda_2^t, \ldots, \lambda_N^t)^t$, λ_i in E^n, defined by

$$\lambda = F_z^{-t} Qz.$$

Then, given u and z_0, the gradient g may be obtained by using the three equations

$$F_z = F_u u + F_0 z_0, \tag{4}$$

$$F_z^t \lambda = Qz , \tag{5}$$

$$g = Ru + F_u^t \lambda. \tag{6}$$

With the notation $g^k = \left(\frac{dJ}{du} \right)^t_{u = u^k}$, the version of the best-step steepest descent method that we use is the following:

Algorithm 1: Let u^1 be given.

Step 0: Set $k = 1$ and compute g^1.

Step 1: If $\|g^k\|^2 \leq \epsilon$ stop; else go to Step 2.

Step 2: Compute α^k to minimize $J(u^k - \alpha^k g^k)$.

Step 3: Compute g^{k+1}.

Step 4: Let $u^{k+1} = u^k - \alpha^k g^k$.

Step 5: Set $k = k + 1$ and go to Step 1.

Note that we compute g^1 in Step 0 and then compute g^{k+1} in Step 3, after the computation of α^k in Step 2. In Section III we present our approach to the computation of g^1, α^k, and g^{k+1} and we show that the computation of α^k and g^{k+1} shares common terms. In Section IV, we discuss efficient parallel procedures for solving block linear recurrences and we use those procedures in Section V to obtain a parallel implementation of Algorithm 1.

III. GRADIENT AND BEST STEP COMPUTATION

We first consider the computation of g^1, which is performed only once in Step 0 of Algorithm 1. As a consequence of Eqs. (4), (5) and (6) we obtain the gradient evaluation technique proposed by Polak [POL71]:

Algorithm 2: Given u^1 and z_0.

Step 1: Compute z^1 such that $F_z z^1 = F_u u^1 + F_0 z_0$.

Step 2: Compute λ^1 such that $F_z^t \lambda^1 = Q z^1$.

Step 3: Compute $g^1 = R u^1 + F_u^t \lambda^1$.

Due to the lower $n \times n$ block bidiagonal structure of F_z, Steps 1 and 2 of Algorithm 2 require procedures for the solution of N stage forward and reverse $n \times n$ block first-order linear recurrences, respectively. Such procedures are presented in next section. Given λ^1, Step 3 computes g^1 by computing each of the N uncoupled components $g_i{}^1 = \left(\frac{dJ}{du_i} \right)^t_{u = u^1}$; thus, Step 3 exhibits linear speedup when executed in parallel.

We now consider the computation of the optimal step length α^k. The cost function is quadratic and therefore a closed form solution for α^k exists. It is clear that

$$J(u) = \frac{1}{2} u^t (R + F_u^t F_z^{-t} Q F_z^{-1} F_u) u ,$$

therefore

$$J(u^k - \alpha g^k) = a\,\alpha^2 + b\,\alpha + c ,$$

where

$$a = \frac{1}{2} < g^k , d^k >,$$

$$b = - < g^k , g^k >,$$

$$c = J(u^k),$$

$$d^k = (R + F_u^t F_z^{-t} Q F_z^{-1} F_u) g^k , \qquad (7)$$

and it follows that the optimal step length α^k is

$$\alpha^k = -\frac{b}{2a} = \frac{< g^k , g^k >}{< g^k , d^k >}.$$

Once d^k is known, the gradient g^{k+1} is easy to evaluate using

$$g^{k+1} = g^k - \alpha^k d^k .$$

The matrix $(R + F_u^t F_z^{-t} Q F_z^{-1} F_u)$ may be precomputed and the quantity d^k may be obtained by performing a single matrix-vector product. However, we show in Section V that a more efficient approach can be obtained by rewriting d^k as

$$d^k = Rg^k + F_u^t F_z^{-t} Q F_z^{-1} F_u g^k ,$$

and using Algorithm 3 below, where we note that instead of requiring F_z^{-1} and F_z^{-t}, we solve linear systems corresponding to F_z and F_z^t.

Algorithm 3: Given g^k.

Step 1: Compute $\mu^k = Rg^k$.

Step 2: Compute $\delta^k = F_u g^k$.

Step 3: Compute w^k such that $F_z w^k = \delta^k$.

Step 4: Compute $\eta^k = Qw^k$.

Step 5: Compute v^k such that $F_z^t v^k = \eta^k$.

Step 6: Compute $\chi^k = F_u^t v^k$.

Step 7: Compute $d^k = \mu^k + \chi^k$.

With the exception of Steps 3 and 5, Algorithm 3 computes d^k by executing a series of matrix-vector products followed by a vector sum. Each of the matrix-vector products consists of N uncoupled block matrix-vector products, thus exhibiting linear speedup when implemented in parallel. However, due to the structure of F_z, Steps 3 and 5 require the solution of N stage forward and reverse $n \times n$ block first-order linear recurrences, respectively. As in the case of

Algorithm 2, this again suggests the need for parallel procedures to solve linear recurrences. Note that the state z^{k+1} and costate λ^{k+1} corresponding to u^{k+1} can be obtained easily from the quantities w^k and v^k computed in Steps 3 and 5 of Algorithm 3, that is,

$$z^{k+1} = z^k - \alpha^k w^k$$

and

$$\lambda^{k+1} = \lambda^k - \alpha^k v^k .$$

IV. PARALLEL PROCEDURES FOR LINEAR BLOCK RECURRENCES

The model of SIMD parallel computation that we use consists of a global parallel memory, p parallel processors, and a control unit, where all processors perform the same operation at each time step (see Fig. 1). We further simplify the model by making the following assumptions:

A1. Each computational operation takes the same amount of time, referred to as a step.

A2. There are no accessing conflicts in global memory.

A3. All initial data resides in global memory.

A4. There is no time required to access global memory.

We now present two parallel procedures to solve forward and reverse N stage $n \times n$ block first-order linear recurrence systems that we use to implement Algorithms 2 and 3. The procedures are blocked versions of the parallel scalar approach given in [MEY86] and are formulated as a function of the number p of processors so that their structure is fixed whenever the number of stages $N \geq \left(\frac{p}{n}\right)^2$.

We first consider the parallel solution of forward recurrences. The forward recurrence problem is: given $n \times n$ matrices A_i, $i = 2, 3, \ldots, N$ and given vectors $\gamma_i \in E^n$, $i = 1, 2, \ldots, N$, find the n vectors z_i such that $z_1 = \gamma_1$ and $z_i = A_i z_{i-1} + \gamma_i$, $i = 2, 3, \ldots, N$. Let

$$\Omega = \begin{cases} \dfrac{N-1}{(p/n)^2-1)} & \text{if } p > n \\ N-1 & \textit{otherwise} \end{cases} \tag{8}$$

and

$$\kappa = \begin{cases} p/n & \text{if } p > n \\ 1 & \textit{otherwise.} \end{cases} \tag{9}$$

For ω in $[0,1,\ldots,\Omega-1]$, define the index sets

$$f_0(\omega) = \{\kappa i + \omega(\kappa^2-1) : i = \kappa-1, \kappa-2, \ldots, 1\}$$

and

$$f_1(\omega) = \{\kappa i + Max(\omega+1, \omega(\kappa^2-1)) : i = \kappa, \kappa-1, \ldots, 1\}.$$

Thus, given $\gamma = (\gamma_1^t, \gamma_2^t, \ldots, \gamma_N^t)^t$, $\gamma_i \in E^n$ and precomputed $A[i+j, j] = A_{i+j} A_{i+j-1} \cdots A_j$, $j \in f_0(\omega)$, i in $[0,1,\ldots,\kappa-1]$, the following procedure solves the forward block recurrence system, where for presentation simplicity, we assume that Ω and κ are integers.

```
1. PROCEDURE FORWARD($N, n, p, \gamma$)
2.   $z_1 := \gamma_1$
3.   FOR $\omega := 0$ TO $\Omega - 1$ DO
4.      FORALL $j \in f_0(\omega)$ DO IN PARALLEL $z_j := \gamma_j$;
5.      FOR $i := 1$ TO $Max(1, \kappa-1)$ DO
6.         FORALL $j \in f_1(\omega)$ DO IN PARALLEL
```

```
7.            $z_{i+j} := A_{i+j} z_{i+j-1} + \gamma_{i+j}$;
8.          END FORALL
9.        END FOR
10.       FORALL $j \in f_0(\omega)$ DO
11.         FOR $i := 0$ TO $\kappa-1$ DO IN PARALLEL
12.           $z_{i+j} := A[i+j, j] z_{j-1} + z_{i+j}$;
13.         END FOR
14.       END FORALL
15.     END FOR
16. END PROCEDURE
```

When $1 \leq p \leq n$, the index set $f_0(\omega)$ is empty and procedure FORWARD reduces to sequentially executing step 7 $N-1$ times, each execution using p processors. When $n < p \leq nN^{1/2}$, procedure FORWARD sequentially solves Ω reduced block recurrence systems in z_i of size $(p/n)^2$, each in parallel. Each reduced system is solved in two phases: the first phase consists of the execution of steps 4 and 5 and computes $(p/n)^2$ partial solutions, in which the first p/n are the actual solutions and the second phase consists of the execution of loop 10 in which the precomputed $n \times n$ block matrices $A[i+j, j]$ are used to update the next p/n partial solutions at each iteration. We assign n processors to perform each of the p/n concurrent executions of steps 7 and 12. The complete solution to the block recurrence system is obtained after executing Ω iterations of loop 3. If Ω is not an integer, then we replace Ω by $\lceil \Omega \rceil$ and simply terminate the computation when z_N is computed and if κ is not an integer, we replace κ by $\lfloor p/n \rfloor$.

We now modify the procedure FORWARD to solve N stage $n \times n$ block reverse linear recurrence systems, where the reverse recurrence problem is: given

$n \times n$ matrices A_i, $i = N-1, N-2, \ldots, 1$ and given vectors $\varsigma_i \in E^n$, $i = N, N-1, \ldots, 1$, find the n vectors λ_i such that $\lambda_N = \varsigma_N$ and $\lambda_i = A_i \lambda_{i+1} + \varsigma_i$, $i = N-1, N-2, \ldots, 1$. Let Ω and κ be defined by Eqs. (8) and (9). For ω in $[0, 1, \ldots, \Omega-1]$, define the index sets

$$r_0(\omega) = \{\kappa i + \omega(\kappa^2-1) : i = \kappa-1, \kappa-2, \ldots, 1\}$$

and

$$r_1(\omega) = \{\kappa i + Max(\omega+1, \omega(\kappa^2-1)) : i = \kappa, \kappa-1, \ldots, 1\}.$$

Given $\varsigma = (\varsigma_1^t, \varsigma_2^t, \ldots, \varsigma_N^t)^t$, $\varsigma_i \in E^n$ and precomputed $A[j+1, j-i+1]^t = A_{j-i+1}^t A_{j-i}^t \cdots A_{j+1}^t$, $j \in r_0(\omega)$, i in $[0, 1, \ldots, \kappa\text{-}1]$, the following procedure solves the reverse block recurrence system, where for presentation simplicity, we again assume that Ω and κ are integers.

```
 1. PROCEDURE REVERSE($N, n, p, \varsigma$)
 2.   $\lambda_N := \varsigma_N$
 3.   FOR $\omega := \Omega - 1$ TO 0 DO
 4.     FORALL $j \in r_0(\omega)$ DO IN PARALLEL $\lambda_j := \varsigma_j$;
 5.     FOR $i := 1$ TO $Max(1, \kappa-1)$ DO
 6.       FORALL $j \in r_1(\omega)$ DO IN PARALLEL
 7.         $\lambda_{j-i} := A_{j-i+1}^t \lambda_{j-i+1} + \varsigma_{j-i}$;
 8.       END FORALL
 9.     END FOR
10.     FORALL $j \in r_0(\omega)$ DO
11.       FOR $i := 0$ TO $\kappa-1$ DO IN PARALLEL
12.         $\lambda_{j-i} := A[j+1, j-i+1]^t \lambda_{j+1} + \lambda_{j-i}$;
13.       END FOR
14.     END FORALL
```

```
15.  END FOR
16. END PROCEDURE
```

We now give the number of steps required to solve either an N stage forward or reverse first-order $n \times n$ block linear recurrence system.

Theorem 1: Given N, n and p such that $p = 1$ or $\frac{p}{n}$ is an integer, the number of parallel steps required to solve a block $n \times n$ first-order linear recurrence system of length N using p processors is

$$T(N,n,p) = \begin{cases} (N-1)\dfrac{2n^2}{p} & \text{if } 1 \le p \le n \\ \left\lceil \dfrac{N-1}{\left(\frac{p}{n}\right)^2 - 1)} \right\rceil 4(p-n) & \text{if } n < p \le nN^{\frac{1}{2}}. \end{cases}$$

It is clear from Theorem 1 that the speedup $S_p = \frac{T_1}{T_p}$ exhibited by the procedures FORWARD and REVERSE is p when $1 \le p \le n$ and $\frac{1}{2}(p+n)$ when $n < p \le nN^{\frac{1}{2}}$ and $\frac{p}{n}$ and $\dfrac{N-1}{\left(\frac{p}{n}\right)^2 - 1}$ are integers. The corresponding efficiency $E_p = \frac{S_p}{p}$ is therefore 1 when $1 \le p \le n$ and $\frac{1}{2} + \frac{n}{2p}$ when $n < p \le nN^{\frac{1}{2}}$ and $\frac{p}{n}$ and $\dfrac{N-1}{\left(\frac{p}{n}\right)^2 - 1}$ are integers. In Figs. 2, 3, 4 and 5 we plot E_p for the values of $n = 8$, 16, 32 and 64 respectively, where the efficiency corresponding to the procedures FORWARD and REVERSE is denoted by the dashed line in each plot. Thus, we see that the efficiency increases with increasing values of n and p and is independent of N.

V. PARALLEL BEST STEP STEEPEST DESCENT

We now use the parallel procedures for the solution of linear recurrences discussed in the previous section to obtain parallel implementations of Algorithms 2 and 3 give the corresponding number of steps required for their execution when p processors are used.

We first give the parallel implementation of Algorithm 2.

```
1. PROCEDURE GRADIENT($z_0$,$u^1$)
2.   FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\gamma_i^1 := B_i u_i^1$;
3.   $\gamma_1^1 := \gamma_1^1 + A_1 z_0$;
4.   $z^1 = $ FORWARD($N, n, p, \gamma^1$);
5.   FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\varsigma_i^1 := Q_i z_i^1$;
6.   $\lambda^1 = $ REVERSE($N, n, p, \varsigma^1$);
7.   FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $g_i^1 := R_i u_i^1 + B_i^t \lambda_i^1$;
8.   RETURN $g^1$;
9. END PROCEDURE
```

Lemma 1: Given z_0, u^1, N, n, m and p such that $p = 1$ or $\frac{p}{n}$ is an integer, the number of steps required by the procedure GRADIENT to compute g^1 using p processors, $1 \le p \le nN^{1/2}$, is

$$T(N,n,m,p) = \begin{cases} \frac{Nn}{p}(6n+2m-2) + \left\lceil \frac{Nm}{p} \right\rceil (2n+2m-1) - \frac{2n^2}{p} & \text{if } 1 \le p \le n \\ \left\lceil \frac{Nn}{p} \right\rceil (2n+2m-2) + \left\lceil \frac{Nm}{p} \right\rceil (2n+2m-1) + \left\lceil \frac{N-1}{\left(\left(\frac{p}{n}\right)^2 - 1\right)} \right\rceil 8(p-n) + 2n & \text{if } n < p \le nN^{1/2}. \end{cases}$$

We next give the parallel implementation of Algorithm 3.

```
1. PROCEDURE DIRECTION($g^k$)
2.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\mu_i^k := R_i g_i^k$;
3.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\delta_i^k := B_i g_i^k$;
4.  $w^k$ = FORWARD($N, n, p, \delta^k$);
5.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\eta_i^k := Q_i w_i^k$;
6.  $v^k$ = REVERSE($N, n, p, \eta^k$);
7.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\chi_i^k := B_i^t v_i^k$;
8.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $d_i^k := \mu_i^k + \chi_i^k$;
9.  RETURN $d^k$;
10. END PROCEDURE
```

Lemma 2: Given g^k, N, n, m and p such that $p = 1$ or $\frac{p}{n}$ is an integer, the number of steps required by procedure DIRECTION to compute d^k using p processors, $1 \leq p \leq nN^{½}$, is

$$T(N,n,m,p) = \begin{cases} \frac{Nn}{p}(6n+2m-2) + \left\lceil \frac{Nm}{p} \right\rceil (2n+2m-1) - \frac{4n^2}{p} & \text{if } 1 \leq p \leq n \\ \left\lceil \frac{Nn}{p} \right\rceil (2n+2m-2) + \left\lceil \frac{Nm}{p} \right\rceil (2n+2m-1) + \left\lceil \frac{N-1}{\left(\left(\frac{p}{n}\right)^2 -1\right)} \right\rceil 8(p-n) & \text{if } n < p \leq nN^{½}. \end{cases}$$

At this point it is interesting to contrast the result of Lemma 2 with the number of steps necessary to obtain d^k based upon the precomputation of the matrix $(R + F_u^t F_z^{-t} Q F_z^{-1} F_u)$. Given p processors, $1 \leq p \leq Nm$, the number of steps required to obtain d^k using Eq. (7) is

$$T(N,m,p) = \left\lceil \frac{Nm}{p} \right\rceil (2Nm - 1).$$

Thus, for p in the range $1 \leq p \leq nN^{½}$, the use of Algorithm 3 to compute d^k results in a smaller number of steps than using the precomputed matrix

$(R + F_u^t F_z^{-t} Q F_z^{-1} F_u)$ and performing a matrix-vector product.

We now embed the parallel procedures GRADIENT and DIRECTION to obtain a parallel implementation of Algorithm 1 and we then give the corresponding number of steps required for one iteration.

```
1. PROCEDURE PSDM($z_0,u^1$)
2.   $k = 1$;
3.   $g^1 =$ GRADIENT($z_0,u^1$);
4.   WHILE $|g^k|^2 > \epsilon$ DO
5.     $d^k =$ DIRECTION($g^k$);
6.     $\alpha^k = <g^k,g^k>/<g^k,d^k>$;
7.     FORALL $i \epsilon \{1,2,...,N\}$ DO IN PARALLEL $g_i^{k+1} := g_i^k - \alpha^k d_i^k$;
8.     FORALL $i \epsilon \{1,2,...,N\}$ DO IN PARALLEL $u_i^{k+1} := u_i^k - \alpha^k g_i^k$;
9.     $k = k + 1$;
10.  END WHILE
11. END PROCEDURE
```

Theorem 2: Given z_0, u^1, N, n, m and p such that $p = 1$ or $\frac{p}{n}$ is an integer, the number of steps required by one iteration of procedure PSDM using p processors, $1 \le p \le nN^{\frac{1}{2}}$, is

$$T(N,n,m,p) = \begin{cases} \left\lceil \frac{Nn}{p} \right\rceil (6n+2m-2) + \left\lceil \frac{Nm}{p} \right\rceil (2n+2m+7) - \frac{4n^2}{p} + 2\log_2 p & \text{if } 1 \le p \le n \\ \left\lceil \frac{Nn}{p} \right\rceil (2n+2m-2) + \left\lceil \frac{Nm}{p} \right\rceil (2n+2m+7) + \left\lceil \frac{N-1}{\left(\left(\frac{p}{n}\right)^2 -1\right)} \right\rceil 8(p-n) + 2\log_2 p & \text{if } n < p \le nN^{\frac{1}{2}}. \end{cases}$$

The speedup S_p and efficiency E_p for procedure PSDM can be obtained directly from Theorem 2 as

$$S_p = \frac{T_1}{T_p},$$

$$E_p = \frac{T_1}{pT_p},$$

where

$$T_1 = Nn(6n + 2m - 2) + Nm(2n + 2m + 7) - 4n^2,$$

$$T_p = \frac{Nn}{p}(2n + 2m - 2) + \frac{Nm}{p}(2n + 2m + 7) + \frac{8(N-1)n^2}{p + n} + 2\log_2 p,$$

and for simplicity, the ceilings have been removed. For the range of values of N, n and m of interest

$$T_1 \approx Nn(6n + 2m - 2) + Nm(2n + 2m + 7),$$

$$T_p \approx \frac{Nn}{p}(2n + 2m - 2) + \frac{Nm}{p}(2n + 2m + 7) + \frac{8Nn^2}{p + n},$$

and thus

$$S_p \approx \frac{p(C + 4n^2)}{C + \frac{8n^2 p}{p + n}},$$

$$E_p \approx \frac{C + 4n^2}{C + \frac{8n^2 p}{p + n}},$$

where

$$C = 2(n + m)^2 - 2n + 7m.$$

Using the fact that

$$\frac{p}{p + n} \leq 1$$

we may find lower bounds for the speedup and efficiency of procedure PSDM, that is

$$S_p \geq \frac{pD}{D + 4n^2},$$

$$E_p \geq \frac{D}{D + 4n^2},$$

where

$$D = 2(n + m)^2 - 2n + 7m + 4n^2.$$

Note that

$$\frac{D}{D + 4n^2} = \frac{6n^2 + E}{10n^2 + E},$$

where

$$E = 2m^2 + 4nm - 2n + 7m,$$

thus $E \geq 0$, and therefore

$$S_p \geq 0.6p,$$

$$E_p \geq 0.6.$$

In Figs. 2, 3, 4 and 5, we plot E_p for the procedure PSDM for the values $n = 8, 16, 32$ and 64 respectively, and in each case use the values $m = 1, 4$ and 8. It is then easy to see that E_p increases with m, and that the efficiency of the procedure PSDM is bounded from below by the efficiency of the procedures FORWARD and REVERSE.

VI. CONCLUSIONS

In this paper a parallel implementation of the best-step steepest descent method has been presented to solve the LQR optimal control problem. The procedure exhibits the desirable property that its structure, and hence parallelism, is determined by the number of available processors. Thus, unlike approaches in which the structure of the procedure changes with problem size, the procedure presented in this paper maintains the same computational and interprocessor communication requirements, independently of the number of stages in the control problem. Furthermore, the procedure has been shown to exhibit an efficiency E_p always greater than 0.6.

The paper's basic approach can be used to produce parallel implementations of more complex gradient based methods. For example, the procedure PSDM may be easily modified to produce the following parallel version of the Fletcher-Reeves conjugate gradient method.

```
1. PROCEDURE PCGM($z_0, u^1$)
2.   $k = 1;$
3.   $\pi^1 = g^1 =$ GRADIENT($z_0, u^1$);
4.   WHILE $|g^k|^2 > \epsilon$ DO
5.     $d^k =$ DIRECTION($\pi^1$);
6.     $\alpha^k = <g^k, \pi^k>/<d^k, \pi^k>;$
7.     FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $g_i^{k+1} := g_i^k - \alpha^k d_i^k;$
8.     FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $u_i^{k+1} := u_i^k - \alpha^k \pi_i^k;$
9.     $\beta^k = <g^{k+1}, g^{k+1}>/<g^k, g^k>;$
10.    FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\pi_i^{k+1} := g_i^{k+1} - \beta^k \pi_i^k;$
11.    $k = k + 1;$
12.  END WHILE
13. END PROCEDURE
```

The procedure PCGM exhibits slightly less speedup than the procedure PSDM due to the additional evaluation of the term β^k. However, this is offset by gaining the improved convergence properties of a conjugate gradient method.

Finally, we note that the procedures presented in this paper may be used to solve discrete optimal control problems which involve nonquadratic cost, nonlinear dynamics and constraints on the states and/or controls. In that case, one needs to use penalty function and gradient projection methods and suitable approximations to cost function and system dynamics.

REFERENCES

[CAR84] Carlson, D.A., and Sugla, B., Time and Processor Efficient Parallel Algorithms for Recurrence Equations and Related Problems, *Proceedings of the 1984 International Conference on Parallel Processing,* August 21-24, 1984, pp. 310-314.

[CHE75] Chen, S.C. and Kuck, D.J., Time and Parallel Processor Bounds for Linear Recurrence Systems, *IEEE Trans. Computers.*, Vol. C-24, No.7, July 1975, pp. 701-717.

[CHE78] Chen, S.C., Kuck, D.J. and Sameh, A.H., Practical Parallel Band Triangular System Solvers, *ACM Trans. on Mathematical Software,* Vol. 4, No. 3, September 1978, pp. 270-277.

[GAJ81] Gajski, D. J., An Algorithm for Solving Linear Recurrence Systems on Parallel and Pipelined Machines, *IEEE Trans. Computers.*, Vol. C-30, No. 3, March 1981, pp. 190-206.

[GRA81] Graham, A., *Kronecker Products and Matrix Calculus : with Applications,* Ellis Horwood Limited, West Sussex, England, 1981.

[KOG73] Kogge, P.M., and Stone, H.S., A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations, *IEEE Trans Computers.*, Vol. C-22, No. 8, August 1973, pp. 786-793.

[KUC76] Kuck, D. J., Parallel Processing of Ordinary Programs, *Advances in Computers.*, Vol. 15, Academic Press, New York, 1976, pp. 119-179.

[LAR73] Larson, R.E. and Edison, T., Parallel Processing Algorithms for the Optimal Control of Nonlinear Dynamic Systems, *IEEE Trans. Computers.*, Vol. C-22, No. 8, August 1973, pp. 777-786.

[LEW86] Lewis, F. L., *Optimal Control,* John Wiley and Sons, New York, N.Y., 1986.

[LUE84] Luenberger, D. G., *Linear and Nonlinear Programming,* Second Edition, Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[MEY73] Meyer, G.G.L., A Segmented Algorithm for Solving a Class of State Constrained Discrete Optimal Control Problems, *Decision and Control Theory Conference,* San Diego, California, 1973, pp. 73-79.

[MEY85] 22yer, G.G.L., and Podrazik, L.J., A Matrix Factorization Approach to the Parallel Solution of First-Order Linear Recurrences, *Proceedings of the 23-rd Annual Allerton Conference on Communication, Control and Computing,* October 2-4, 1985, pp. 243-250.

[MEY86] Meyer, G.G.L., and Podrazik, L.J., A Parallel First-Order Linear Recurrence Solver, *Proceedings of the 20-th Annual Conference Information Sciences & Systems,* Princeton, New Jersey, March 19-21, 1986.

[POL71] Polak, E., *Computational Methods in Optimization. A Unified Approach,* Academic Press, New York and London, 1971.

[SAM77] Sameh, A.H., and Brent, R.P., Solving Triangular Systems on a Parallel Computer, *SIAM J. Numer. Anal.*, Vol. 14, No. 6, December 1977, pp. 1101-1113.

[SCH81] Scheel, C. and McInnis, B., Parallel Processing of Optimal Control Problems by Dynamic Programming, *Information Sciences*, Vol. 25, No. 2, November 1981, pp. 85-114.

[TRA80] Travassos, R. and Kaufman, H., Parallel Algorithms for Solving Nonlinear Two-Point Boundary-Value Problems Which Arise in Optimal Control, *Journal of Optimization Theory and Applications*, Vol. 30, No. 1, January 1980, pp. 53-71.

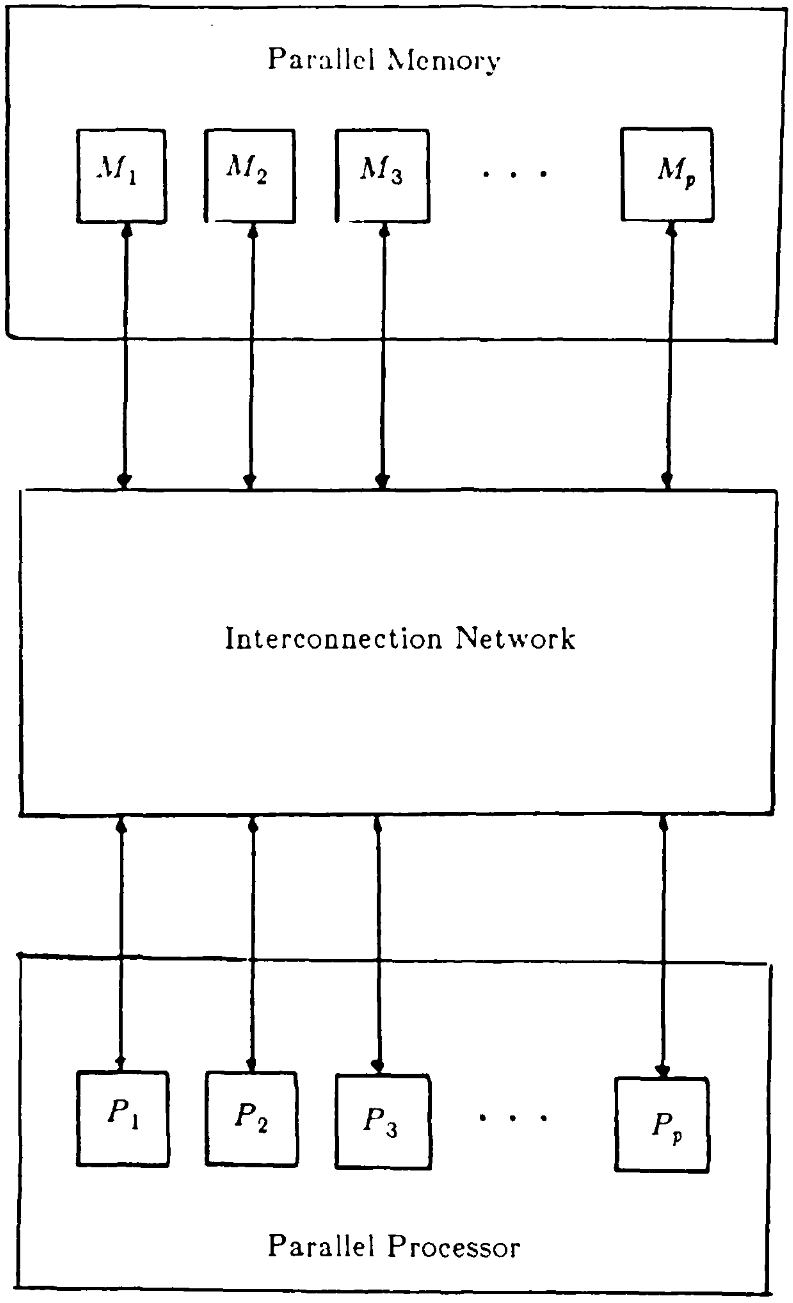


Figure 1. The Abstract Parallel Computational Model

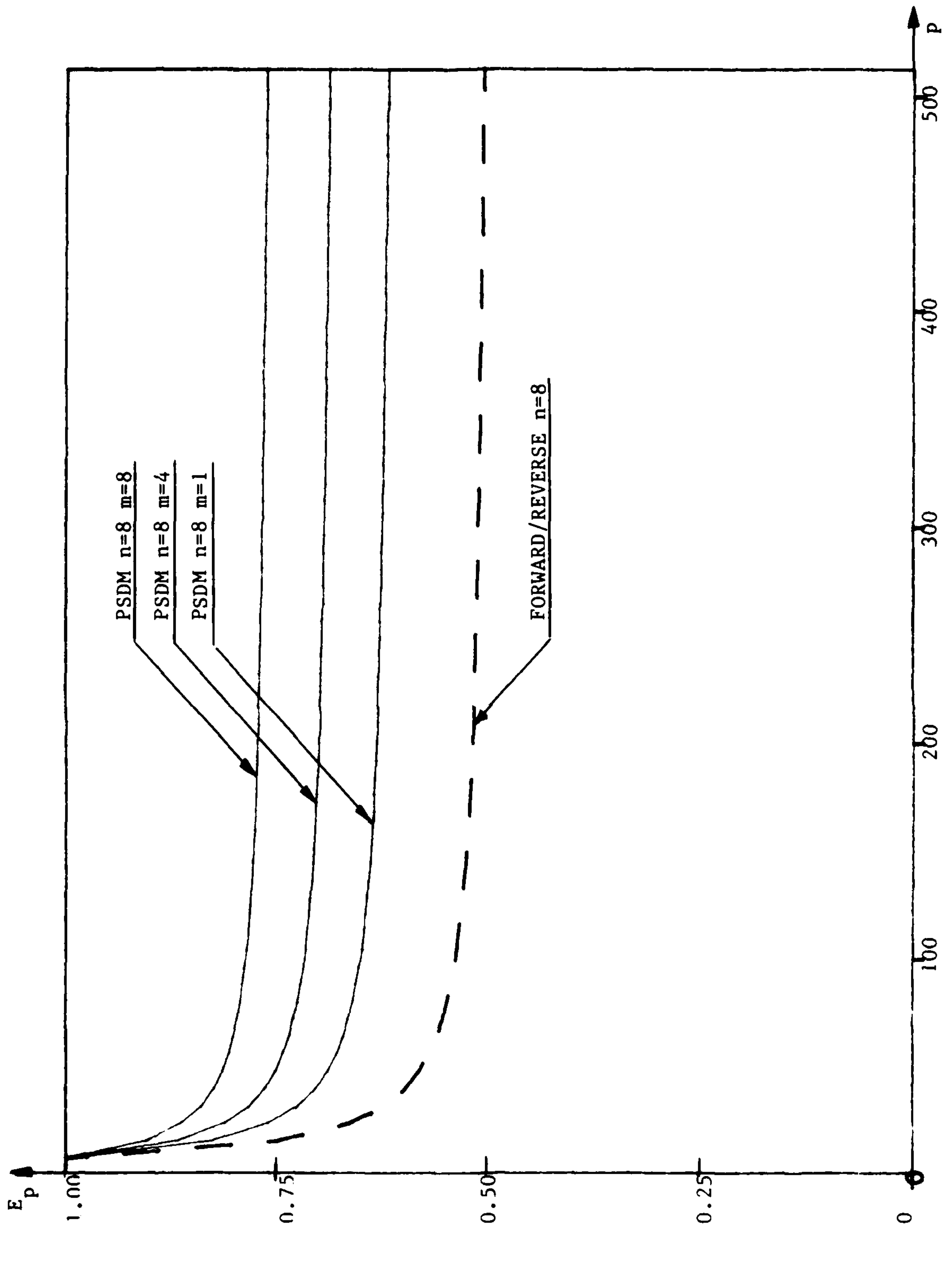


Figure 2. Efficiency of procedures FORWARD, REVERSE and PSDM

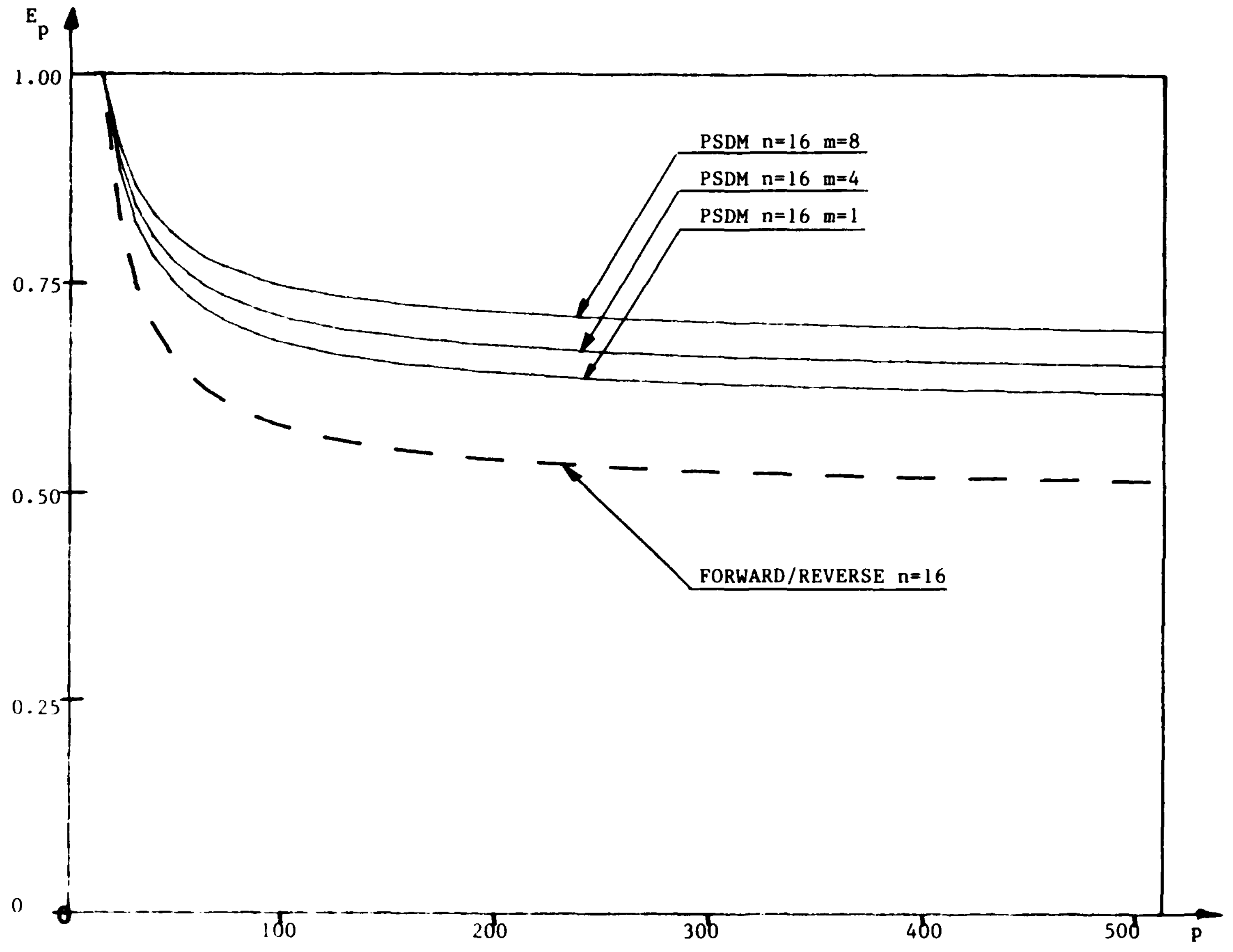


Figure 3. Efficiency of procedures FORWARD, REVERSE and PSDM

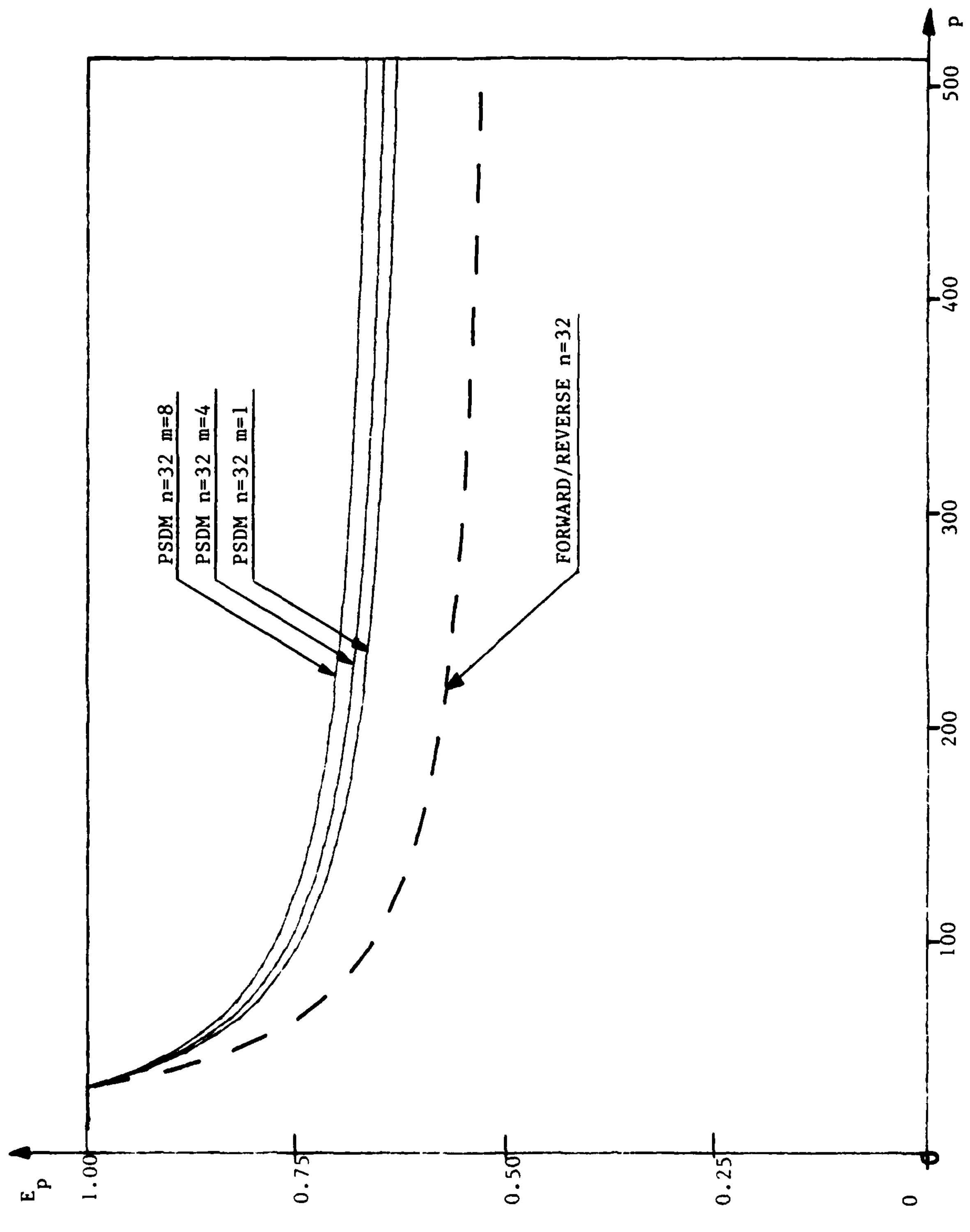


Figure 4. Efficiency of procedures FORWARD, REVERSE and PSDM

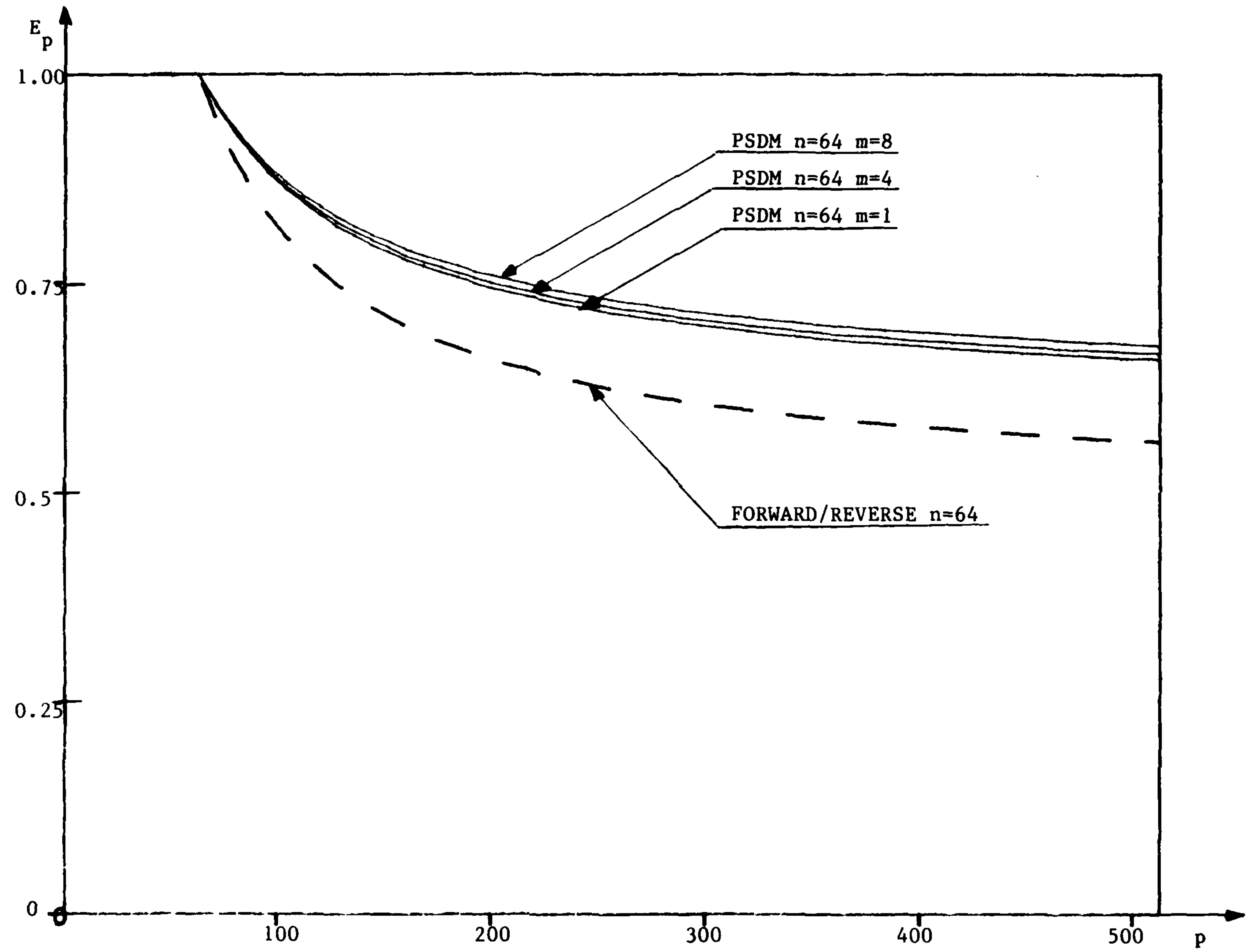


Figure 5. Efficiency of procedures FORWARD, REVERSE and PSDM